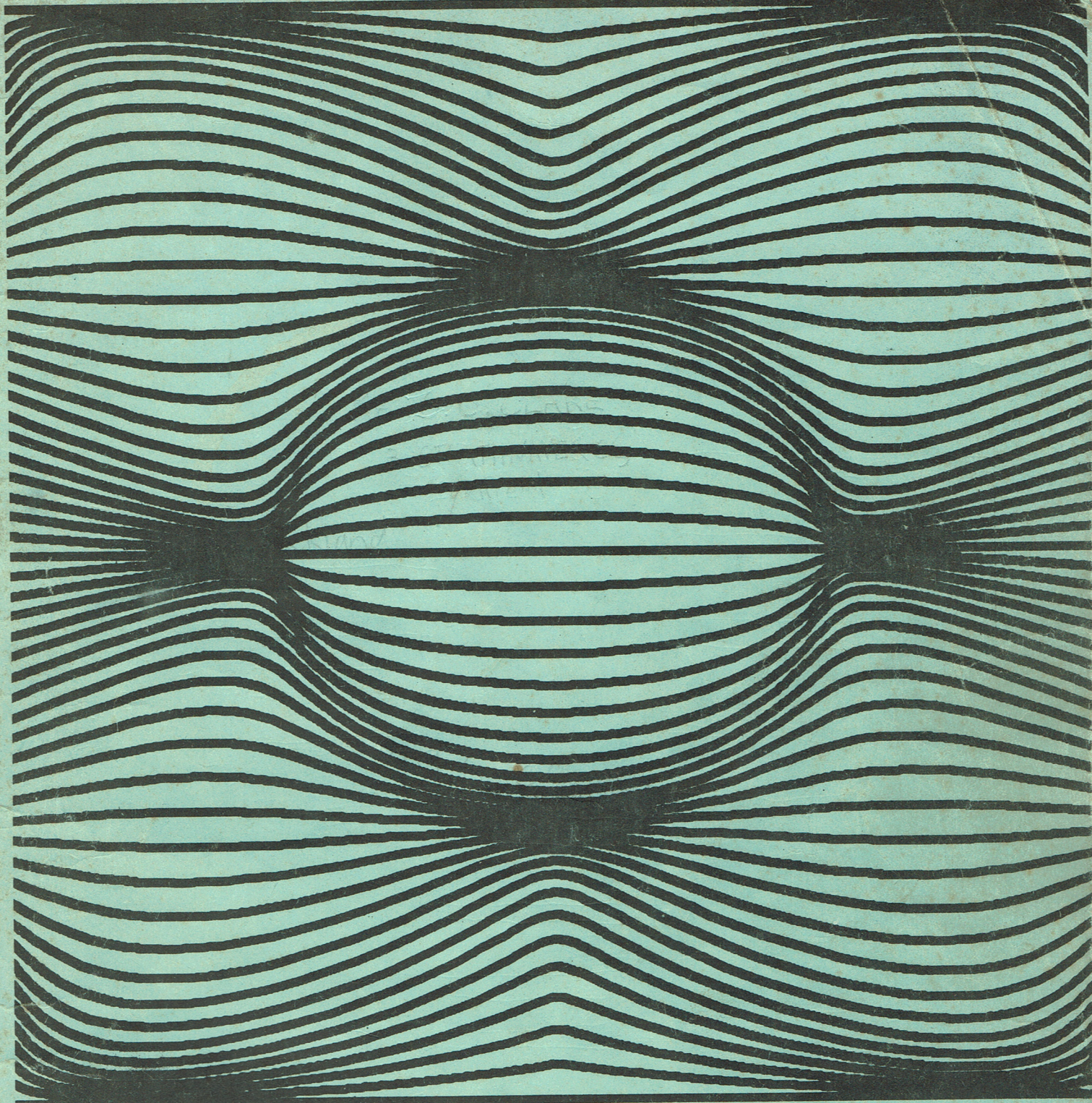




CHIPPOS

MICHAEL J BAUER

Greg. Unre

PREFACE

C H I P O S

Software Manual for the D.R.E.A.M-6800
VIDEO COMPUTER.

by Michael J. BAUER

published by

DREAMWARE
P.O. Box 343
Belmont VIC 3216
Australia

(M.J. Bauer)

Deakin University Printery 1979

PREFACE

This booklet is a programmers' reference manual for a home/school computer known as the DREAM-6800: that's Domestic Recreational & Educational Adaptive Microcomputer; based on the world's most powerful truly 8-bit microprocessor ... the Motorola M6800! Details of the DREAM-6800 system are to be found in 'Electronics Australia' magazine; May, June, July and August issues, 1979. The information contained in this manual is supplementary to these articles. The reader is referred to the Motorola M6800 Applications Handbook (for advanced users), and (for beginners) the text: "Basic Microprocessors and the 6800" by Ron Bishop (Hayden, 1979) available from Rank Industries (Aust) Pty Ltd and Motorola Semiconductor Inc. An in-depth treatment of CHIP-8 programming can be found in an article by the originator Joe Weisbecker in "An Easy Programming System" (BYTE, Dec. '78).

The philosophy of the DREAM-6800 video computer is simplicity. There is much that can be done with such a simple system, without any add-ons. Many constructors will be tempted to add 16K RAM boards, ASCII keyboards, printers... you name it. But the real purpose is to stretch the imagination to see what can be done with the system as it stands: with one-K RAM, CHIPOS EPROM, 64x32 dot video display, HEX keypad, timer, bleeper and cassette. Only when you've exhausted the capabilities of the minimum system do you have an excuse for adding extra memory, relays, flashing lights, sound generators.....

Readers who develop interesting programs and applications for the DREAM-6800 are encouraged to forward their ideas to DREAMWARE (an M6800 software house). If the response is good, these ideas will be printed and distributed to subscribers of 'DREAMer' (the DREAM-6800 User Group journal, TO BE ANNOUNCED.)

(M.J. Bauer)

CHIPOS Software Design and System Integration

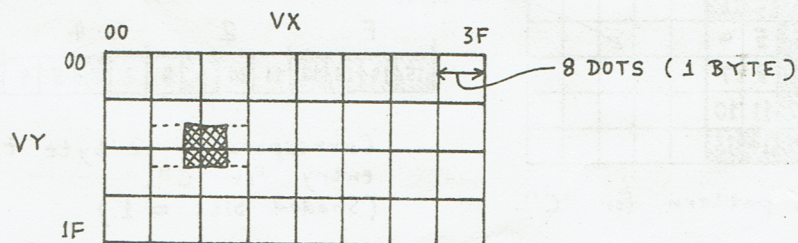
CHIPOS (Compact Hexadecimal Interpretive Programming and Operating System) has been written as an integrated package of subroutines, for two purposes:

1. To allow system routines to be shared by the CHIP-8 Interpreter, monitor, and utility programs; and,
2. To make the system routines available to the machine-code programmer.

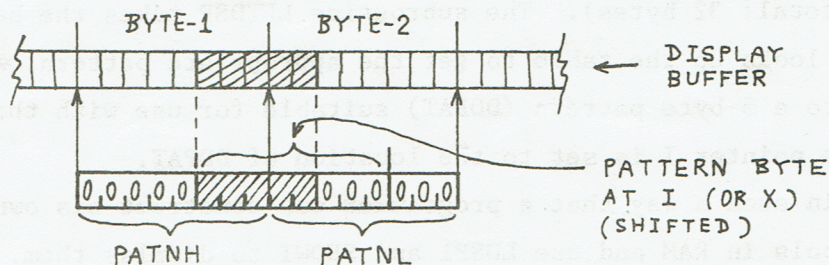
These routines, although compact and efficient, are reasonably well "structured" for ease of use by the caller. Thus, a lot of memory space and effort can be saved by utilising CHIPOS subroutines for such tasks as: Displaying symbols (e.g. hex digits), Inputting from the keypad, Performing serial I/O, Timing, etc. Before examining each subroutine in detail, a few words of explanation about the trickier routines might be in order.

How the Display Driver Routines work:

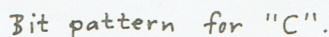
The subroutines SHOW, SHOWI, SHOWX, SHOWUP, DISLOC etc, perform the task of writing a symbol pattern from anywhere in memory into the Display Refresh Buffer (0100-0200) at a specified location given by a coordinate pair (VX and VY). This task is made awkward by the fact that the symbol will usually overlap 2 horizontally adjacent bytes in the buffer, thus:



Consider the top 2 bytes only, in detail:



The system needs a way of storing patterns for the symbols: 0123456789 and ABCDEF (i.e. hex digits). The most straightforward way would be to use 5 bytes for each symbol. This would require $16 \times 5 = 80$ bytes of storage, most of which is wasted (only the first 3 bits are needed). To save space, a more compact method was devised. Since 15 bits are needed for each symbol, only 2 bytes are necessary to store each digit pattern:-



The "T.V. Typewriter" game uses this technique.

The Interrupt Structure

CHIPOS initializes the system so that only the RTC (timer pulses) can cause interrupts. The interrupt service routine (ISR) at C3E9 merely decrements 2 scratchpad parameters (TIME and TONE) which is a requirement of the CHIP-8 Interpreter and the BLEEP routine. Should the user require a different interrupt structure, the program must store the address of the ISR into locations 0000-0001. (N.B: GETKEY calls BLEEP which needs the DEC TONE and TST PIAB instructions in the ISR.)

Example: A certain program requires a "live" keypad, i.e. a key depression will interrupt the execution of the main program and store the inputted digit(s) in some memory location(s).

The main program must first write the address of the ISR into the IRQ vector (0000-1), and it must set up the PIA so that CA1 causes an interrupt. The ISR may call PAINZ and KEYINP to input a digit, then store it, then re-initialize the PIA before the RTI.

WARNING! The main program and the ISR must not share a common subroutine (e.g. SHOW) unless IRQ is disabled prior to the JSR, because the mainline might be using the subroutine at the time the IRQ occurred, and the ISR will then corrupt the scratchpad bytes that the mainline was using!

Software interrupts are easier to use. If you have a subroutine in your program which is frequently accessed, or uses nearly all the MPU registers, why not use SWI/RTI instead of JSR/RTS ? Note that SWI stacks ALL registers and status, not just the PC. Put this subroutine (ISR) at 0080, or put a JMP at 0080. SWI is also very useful for diagnostic purposes, by using SWIs as breakpoints (which can be replaced with NOPs when the program works). Refer to the "DREAMBUG" routine given elsewhere.

What about NMI ? Forget it !! If you can't do it without NMI, it's not worth doing. (The ONLY sensible use for NMI is as a power-fail battery backup service routine, which is what was put there for.) Interrupts, in general, are avoided by designers unless the advantages of using them are paramount.

CHIPOS SUBROUTINES (& Calling Sequences)

ERASE = C079

Function: Clears the display buffer.

Affected parameters: A, X, DISBUF

Calling sequence:

```
BD C079      JSR  ERASE
```

FILL = C07D

Fill part or all of display buffer with constant byte.

Input: A = byte to show

X = starting location; e.g. 0100 for whole screen

Affected: X, DISBUF (stops at 0200)

Call:

```
CE xxxx      LDX  #$xxxx    start locn
86 kk        LDA  A  #$kk     load constant
BD C07D      JSR  FILL
```

RANDOM = C132

Generate a pseudorandom byte.

Input: (optional: initialize RNDX+1 & RND)

Output: A, RND

Affected: RNDX, X

Call:

```
BD C132      JSR  RANDOM
```

LETDSP = C193

Called prior to SHOWI to display a hex digit as a 3x5 symbol.

Input: A = digit to be displayed

Output: I, DDPAT

Affected: A, B, X, PATNH, PATNL

Call:

```
LDA A  xxx    load digit
BD C193      JSR  LETDSP    setup I and DDPAT
C6 05        LDA  B  #5     show 5-byte pattern
BD C224      JSR  SHOWI
```

DECEQ = C1E0

Store 3-digit BCD equivalent of A at X, X+1, X+2.

Input: A = unsigned binary no.

Output: Memory at X, X+1, X+2 (3 bytes)

Affected: A, B, X (=X+3)

Call:

```
LDA A  xxx    byte to convert
LDX      LDX   loc'n for result
BD C1E0      JSR  DECEQ
```

SHOWI = C224, SHOWX = C226

SHOWI (-X) displays an N-byte symbol in memory pointed at by I (X).

Display dots are XORed with existing dots.

Horizontal and vertical wrap-round occurs across borders.

If N = 0 (B-reg.), then N = 16 is assumed.

Input: I (or X) = pointer; B = N (no. of bytes);

VX, VY = screen coordinates for pattern.

Output: DISBUF, VF (=01 if overlap)

Affected: A, B, X (=X+N), VY (=VY+N), VF, BLOC, ZHI, PATNH-L

Call: Initialize: VX, VY, VF=00 (option), I or X, B (=N), then:-

```
BD C224/6    JSR  SHOWI/X
```


DISLOC = C275

Computes the address of the display byte at coords (B, VY).

Input: BLOC = \$01 (DISBUF MSB),
B = x-coordinate; VY = y-coord.

Output: X, BLOC+1 (adrs of req'd byte)

Affected: A, B.

Note: The dot position within the byte is determined from
VX, (LS 3 bits).

PAINZ = C287

Initializes the keypad port; clears PIA flags; disables CA1 IRQ.

Affected: B, X (=PIA adrs), PIAA, PIAA+1

KEYINP = C297

Decodes the hex keypad, after a de-bounce delay of 3.33 msec.

A flag, BADRED, is set to \$0F if a bad read occurred, or if no key
was down, else BADRED = 00.

Returned: A = hex keycode; KEYCOD (=A); BADRED.

Affected: A, B, X (= #PIAA).

GETKEY = C2C4

Waits for a key to be pressed, then, if it's a HEX key, calls KEYINP;
if FN key, returns with A = \$8C (negative). A valid keystroke is
acknowledged with a BLEEP.

Returned: A = hex or 8C (FN). (See also KEYINP, BLEEP, RTC.)

Affected: B, KEYCOD, BADRED, TONE, XTEMP (X is saved)

BLEEP = C2DF

Generates a 2400 Hz tone in speaker for approx 80 msec.

Affected: B, TONE

BTON = C2E5

Generate a variable length tone at either 2400 Hz or 1200 Hz.

Enter: B = \$40 (1200Hz) or \$41 (2400Hz)

TONE = duration x 20 msec (eg: TONE=05 for 100ms)

DEL333 = C2F3; (DEL167 = C2F5)

Delay for 3.33 msec (1.67 msec), assuming display/DMA is off.

Affected: nil.

PBINZ = C2FE

Initialize: serial I/O (tape), tone, RTC timer and display/DMA.

Input: A = \$36 RTC off, DMA off

A = \$3E RTC off, DMA on

A = \$37 RTC on, DMA off

A = \$3F RTC on, DMA on

Affected: B, X (= #PIAB), PIAB, PIAB+1

INBYT = C310

Inputs a byte from the serial data input line PB7, in standard
300 Baud async format (1 start bit, 8 data bits, 1 or more stop).

Note: Display/DMA must be disabled (see PBINZ, above).

Affected: A (returned byte), B, XTEMP.

OUTBYT = C32B

Outputs a byte (A) to the serial data output line PBO at 300 Bd.

Note: Display/DMA must be disabled first.

Affected: B, XTEMP. (A and X are preserved)

BYTIN = C390

Accepts 2 hex digits from the keypad and builds a composite byte (A).

Returned: A.

Affected: A, B, ATEMP, (see also: GETKEY).

START = C360

Monitor entry point.

Terminate a machine-code program with: JMP \$C360.

SHODAT = C3C8

Displays a byte (2 hex digits) in memory, pointed at by X, on the bottom part of the screen.

Enter: X = loc'n of byte to show;

VX = horiz. cursor position.

Note: VY must be set to \$1A prior to the first call to SHODAT.

The display buffer from \$01C8 onwards must be cleared also.

SHOBYT = C3CA

Same as SHODAT except the byte to be shown is in the A-reg.

Affected: A, B, VX (=VX+8); same for SHODAT.

DIGOUT = C3D2

Similar to SHOBYT, but only one digit is displayed (LS 4 bits).

CURSR = C3DC

Used in conjunction with SHODAT, SHOBYT & DIGOUT. Moves "cursor" position to the right 4 dots; ie. adds 4 to VX.

Enter: nil.

Affected: A, VX (=VX+4), VY (= \$1A), X (=XTEMP)

CURS1 = C3E0

Used with SHODAT, SHOBYT, DIGOUT, to reset "cursor" position.

Enter: A = horiz. cursor position (\$00 to \$3C).

Affected: VX (=A), VY (= \$1A), X (=XTEMP).

Note: The above routines will only work on the bottom row of the screen, i.e. VY is fixed at \$1A.

Scratchpad parameter addresses for the above subroutines:

DDPAT	0008	RND	000D	ATEMP	000F
XTEMP	0012	ZHI	0014	ZLO	0015
KEYCOD	0017	BADRED	0018	BLOC	001C
PATNH	001E	PATNL	001F	TIME	0020
tone	0021	I	0026	RNDX	002C
VX	002E	VY	002F	VF	003F
DISBUF	0100	PIAA	8010	PIAB	8012

%%
%%

M6800 MICROPROCESSOR CROSS-ASSEMBLER
[ASM68 REV.II (FORTRAN) BY M.J. BAUER, 1978]
DEAKIN UNIVERSITY DEC-SYSTEM-20/50

%%
%%

CHIPOS

COMPACT HEX. INTERPRETIVE PROGRAMMING AND
OPERATING SYSTEM.

CHIP8 LANGUAGE INTERPRETER
PLUS OPERATING SYSTEM
FOR 6800 COMPUTERS WITH 1K ROM + 1K RAM

COMPATIBLE WITH RCA COSMAC VIP
[SOURCE BEGINS AT LOC 0200]

[N.B:-

- (1) UPON RELOCATION, THE DATA AT LOC \$C133
MUST BE CHANGED ACCORDINGLY (SEE RANDOM);
- (2) IF DISPLAY AREA MOVED, THE DATA AT LOC \$C227
IS CHANGED TO HIGH-ORDER BYTE OF BUFFER ADRS.]

SCRATCHPAD RAM ASSIGNMENTS (PAGE 0)

0000	IRQV	EQU	\$0000	INTERRUPT VECTOR
0002	BEGA	EQU	\$0002	BEGIN ADRS FOR LOAD/DUMP
0004	ENDA	EQU	\$0004	ENDING ADRS FOR LOAD/DUMP
0006	ADRS	EQU	\$0006	ADRS FOR GO AND MEMOD
0008	DDPAT	EQU	\$0008	DIGIT PATTERN TEMP (5 BYTES)
000D	RND	EQU	\$000D	RANDOM BYTE (SEED)
000E	N	EQU	\$000E	TEMP
000F	ATEMP	EQU	\$000F	TEMP
0012	XTEMP	EQU	\$0012	2-BYTE SAVE FOR X, SP
0014	ZHI	EQU	\$0014	TEMP ADRS
0015	ZLO	EQU	\$0015	
0017	KEYCOD	EQU	\$0017	KEYCODE TEMP
0018	BADRED	EQU	\$0018	KEY BAD-READ FLAG
001C	BLOC	EQU	\$001C	DISPLAY POINTER (BYTE LOC'N)
001E	PATNH	EQU	\$001E	PATTERN TEMP
001F	PATNL	EQU	\$001F	
0020	TIME	EQU	\$0020	RTC TIMER VALUE
0021	TONE	EQU	\$0021	DURATION COUNT FOR TONE
0022	PPC	EQU	\$0022	PSEUDO PRGM-COUNTER
0024	PSP	EQU	\$0024	PSEUDO STACK-PTR
0026	I	EQU	\$0026	CHIP8 MEMORY POINTER
0028	PIR	EQU	\$0028	PSEUDO INST-REG
002A	VXLOC	EQU	\$002A	POINTS TO VX
002C	RNDX	EQU	\$002C	RANDOM POINTER
002E	VX	EQU	\$002E	VARIABLE X [ALSO X-COORD]
002F	VY	EQU	\$002F	VARIABLE Y [ALSO Y-COORD]

CHIP8 VARIABLES (TABLE)

0030	VO	EQU	\$0030	
003F	VF	EQU	\$003F	
	CHIP8 SUBROUTINE STACK			
005F	STACK	EQU	\$005F	

	*** OPERATING-SYSTEM STACK ***			
007F	STOP	EQU	\$007F	STACK TOP (MONITOR)

*** CHIP8 GRAPHIC DISPLAY AREA:
(1/4 K RAM BLOCK MAPPED ONTO T.V. SCREEN BY DMA,
IN FORMAT 64H*32V DOTS)

0100	DISBUF	EQU	\$0100	DISPLAY BUFFER AREA
0200	ENDBUF	EQU	\$0200	
8010	PIAA	EQU	\$8010	PORT-A FOR KEYPAD
8012	PIAB	EQU	\$8012	PORT-B FOR TAPE , RTC, TONE

*** CHIP8 INTERPRETER MAINLINE ***

C000		ORG	\$C000	
C000 8D 77	CHIP8	BSR	ERASE	NORMAL ENTRY POINT
C002 CE 0200		LDX	##0200	RESET PSEUDO-PC
C005 DF 22		STX	PPC	
C007 CE 005F		LDX	#STACK	RESET STACK PTR
C00A DF 24		STX	PSP	
C00C DE 22	FETCH	LDX	PPC	POINT TO NEXT INSTR
C00E EE 00		LDX	0,X	COPY TO PIR
C010 DF 28		STX	PIR	
C012 DF 14		STX	ZHI	SAVE ADRS (MMM)
C014 BD C0D0		JSR	SKIP2	BUMP PRGM-CTR
C017 96 14		LDA A	ZHI	MASK OFF ADRS
C019 84 0F		AND A	##0F	
C01B 97 14		STA A	ZHI	
C01D 8D 21		BSR	FINDV	EXTRACT VX ALSO
C01F 97 2E		STA A	VX	STASH VX
C021 DF 2A		STX	VXLOC	SAVE LOCATION OF VX
C023 96 29		LDA A	PIR+1	FIND Y
C025 44		LSR A		
C026 44		LSR A		
C027 44		LSR A		
C028 44		LSR A		
C029 8D 15		BSR	FINDV	EXTRACT VY
C02B 97 2F		STA A	VY	STASH VY
C02D CE C048	EXEC	LDX	#JUMTAB-2	POINT TO JUMP TABLE
C030 96 28		LDA A	PIR	EXTRACT MSD
C032 84 F0		AND A	##F0	
C034 08	EXE1	INX		FIND ROUTINE ADRS
C035 08		INX		
C036 80 10		SUB A	##10	
C038 24 FA		BCC	EXE1	BRANCH IF HIGHER OR SAME
C03A EE 00		LDX	0,X	LOAD ROUTINE ADRS
C03C AD 00		JSR	0,X	PERFORM ROUTINE
C03E 20 CC		BRA	FETCH	NEXT INSTR....
C040 CE 002F	FINDV	LDX	#V0-1	POINT TO VARIABLES TABLE
C043 08	FIND1	INX		FIND LOCN VX
C044 4A		DEC A		
C045 2A FC		BPL	FIND1	
C047 A6 00		LDA A	0,X	FETCH VX FROM TABLE
C049 39		RTS		

JUMP TABLE (ROUTINE ADDRESSES):

C04A C0 6A	JUMTAB	FDB	EXCALL	ERASE, RET, CALL, NOP
C04C C0 A2		FDB	GOTO	GOTO MMM
C04E C0 AC		FDB	DOSUB	DO MMM
C050 C0 BA		FDB	SKFEK	SKF VX=KK
C052 C0 C1		FDB	SKFNK	SKF VX#KK
C054 C0 C8		FDB	SKFEV	SKF VX=VY
C056 C0 EE		FDB	LETK	VX=KK
C058 C0 F2		FDB	LETVK	VX=VX+KK
C05A C0 FE		FDB	LETVV	VX=[VX][+--&!J]VY
C05C C0 CC		FDB	SKFNV	SKF VX#VY
C05E C0 A7		FDB	LETI	I=MMM

C060	C0	97	FDB	GOTOV	GOTO MMM+VO
C062	C0	F8	FDB	RANDV	VX=RND.KK
C064	C2	1F	FDB	SHOW	SHOW N@VX,VY
C066	C0	D7	FDB	SKFKEY	SKF VX[=#]KEY
C068	C1	5F	FDB	MISC	(MINOR JUMP TABL)

ERASE, RETURN, CALL (MLS), OR NOP INTRN:					
C06A	D6	28	EXCALL	LDA B PIR	GET INSTR REG
C06C	26	25		BNE CALL	
C06E	96	29		LDA A PIR+1	
C070	81	E0		CMP A #\$E0	
C072	27	05		BEQ ERASE	
C074	81	EE		CMP A #\$EE	
C076	27	0E		BEQ RETDO	
C078	39			RTS	NOP, FETCH
C079	4F		ERASE	CLR A	WRITE ZEROS TO SCREEN
C07A	CE	0100		LDX #DISBUF	POINT TO DISPLAY BUFF
C07D	A7	00	FILL	STA A 0,X	FILL SCREEN WITH ACC-A
C07F	08			INX	
C080	8C	0200		CPX #ENDBUF	DONE?
C083	26	F8		BNE FILL	
C085	39			RTS	
C086	30		RETDO	TSX	SAVE REAL SP
C087	9E	24		LDS PSP	
C089	32			PUL A	
C08A	97	22		STA A PPC	PULL PPC
C08C	32			PUL A	
C08D	97	23		STA A PPC+1	
C08F	9F	24		STS PSP	SAVE CHIP8 SP
C091	35			TXS	RESTORE SP
C092	39			RTS	
C093	DE	14	CALL	LDX ZHI	GET OPRND ADRS (MMM)
C095	6E	00		JMP 0,X	PERFORM MLS
C097	96	30	GOTOV	LDA A VO	16-BIT ADD VO TO ADRS
C099	5F			CLR B	
C09A	9B	15		ADD A ZLO	
C09C	97	15		STA A ZLO	
C09E	D9	14		ADC B ZHI	
COA0	D7	14		STA B ZHI	
COA2	DE	14	GOTO	LDX ZHI	MOVE ADRS TO PPC
COA4	DF	22		STX PPC	
COA6	39			RTS	FETCH
COA7	DE	14	LETI	LDX ZHI	MOVE ADRS TO MI PTR
COA9	DF	26		STX I	
COAB	39			RTS	FETCH
COAC	30		DOSUB	TSX	SAVE SP
COAD	9E	24		LDS PSP	
COAF	96	23		LDA A PPC+1	PUSH PPC
COB1	36			PSH A	
COB2	96	22		LDA A PPC	
COB4	36			PSH A	
COB5	9F	24		STS PSP	SAVE CHIP SP
COB7	35			TXS	RESTORE REAL SP
COB8	20	E8		BRA GOTO	JUMP TO ADRS (MMM)

CONDITIONAL SKIP ROUTINES:

COBA 96 29	SKFEK	LDA A	PIR+1	GET KK
COBC 91 2E	SKFEQ	CMP A	VX	
COBE 27 10		BEQ	SKIP2	
COC0 39		RTS		
COC1 96 29	SKFNK	LDA A	PIR+1	GET KK
COC3 91 2E	SKFNE	CMP A	VX	
COC5 26 09		BNE	SKIP2	
COC7 39		RTS		
COC8 96 2F	SKFEV	LDA A	VY	GET VY
COCA 20 F0		BRA	SKFEQ	
COCC 96 2F	SKFNV	LDA A	VY	
COCE 20 F3		BRA	SKFNE	
COD0 DE 22	SKIP2	LDX	PPC	ADD 2 TO PPC
COD2 08		INX		
COD3 08		INX		
COD4 DF 22		STX	PPC	
COD6 39		RTS		
COD7 BD C297	SKFKEY	JSR	KEYINP	INTERROGATE KEYBOARD
CODA 7D 0018		TST	BADRED	KEY DOWN?
CODD 27 07		BEQ	SKFK1	
CODF C6 A1		LDA B	##A1	WHAT INSTRN?
COE1 D1 29		CMP B	PIR+1	SKF VX#KEY
COE3 27 EB		BEQ	SKIP2	
COE5 39		RTS		NO KEY; GO FETCH
COE6 C6 9E	SKFK1	LDA B	##9E	
COE8 D1 29		CMP B	PIR+1	WHAT INSTRN?
COEA 27 D0		BEQ	SKFEQ	
COEC 20 D5		BRA	SKFNE	

ARITHMETIC/LOGIC ROUTINES:

COEE 96 29	LETK	LDA A	PIR+1	GET KK
COF0 20 3B		BRA	PUTVX	
COF2 96 29	LETVK	LDA A	PIR+1	
COF4 9B 2E		ADD A	VX	
COF6 20 35		BRA	PUTVX	
COF8 8D 38	RANDV	BSR	RANDOM	GET RANDOM BYTE
COFA 94 29		AND A	PIR+1	
COFC 20 2F		BRA	PUTVX	
COFE 96 2E	LETVV	LDA A	VX	
C100 D6 29		LDA B	PIR+1	
C102 C4 0F		AND B	##0F	EXTRACT N
C104 26 02		BNE	LETV1	
C106 96 2F		LDA A	VY	VX=VY
C108 5A	LETV1	DEC B		
C109 26 02		BNE	LETV2	
C10B 9A 2F		ORA A	VY	VX=VX!VY (OR)
C10D 5A	LETV2	DEC B		
C10E 26 02		BNE	LETV4	
C110 94 2F		AND A	VY	VX=VX.VY

C112 5A	LETV4	DEC B		
C113 5A		DEC B		
C114 26 0A		BNE	LETV5	
C116 7F 003F		CLR	VF	VF=0
C119 9B 2F		ADD A	VY	VX=VX+VY
C11B 24 03		BCC	LETV5	RESULT < 256 ?
C11D 7C 003F		INC	VF	VF=1 (OVERFLOW)
C120 5A	LETV5	DEC B		
C121 26 0A		BNE	PUTVX	
C123 7F 003F		CLR	VF	VF=0
C126 90 2F		SUB A	VY	VX=VX-VY
C128 25 03		BCS	PUTVX	VX<VY? (UNSIGNED)
C12A 7C 003F		INC	VF	NO, PUT VF=1
C12D DE 2A	PUTVX	LDX	VXLOC	REPLACE VX
C12F A7 00		STA A	0,X	
C131 39		RTS		

*** RANDOM BYTE GENERATOR ***

C132 86 C0	RANDOM	LDA A	##C0	** HIGH-ORDER BYTE OF RNDX =
C134 97 2C		STA A	RNDX	= MSB OF CHIP8 START ADRS.
C136 7C 002D		INC	RNDX+1	
C139 DE 2C		LDX	RNDX	POINT TO NEXT PROGRAM BYTE
C13B 96 0D		LDA A	RND	GET SEED (LAST VALUE)
C13D AB 00		ADD A	0,X	MANGLE IT
C13F A8 FF		EOR A	\$FF,X	
C141 97 0D		STA A	RND	STASH IT
C143 39		RTS		

JUMP TABLE FOR MISCELLANEOUS INSTRNS [FXZZ]

C144 07	MINJMP	FCB	\$07	VX=TIME
C145 C1 79		FDB	VTIME	
C147 0A		FCB	\$0A	VX=KEY
C148 C1 7D		FDB	VKEY	
C14A 15		FCB	\$15	TIME=VX
C14B C1 82		FDB	TIMEV	
C14D 18		FCB	\$18	TONE=VX
C14E C1 85		FDB	TONEV	
C150 1E		FCB	\$1E	I=I+VX
C151 C1 89		FDB	LETIV	
C153 29		FCB	\$29	I=DSPL,VX
C154 C1 93		FDB	LETDSP	
C156 33		FCB	\$33	MI=DEQ,VX
C157 C1 DE		FDB	LETDEQ	
C159 55		FCB	\$55	MI=V0:VX
C15A C1 FA		FDB	STORV	
C15C 65		FCB	\$65	V0:VX=MI
C15D C2 04		FDB	LOADV	

C15F CE C144	MISC	LDX	#MINJMP	POINT TO TABLE
C162 C6 09		LDA B	#9	DO 9 TIMES....
C164 A6 00	MIS1	LDA A	0,X	GET TABLE OPCODE
C166 91 29		CMP A	PIR+1	
C168 27 09		BEQ	MIS2	
C16A 08		INX		
C16B 08		INX		
C16C 08		INX		
C16D 5A		DEC B		
C16E 26 F4		BNE	MIS1	
C170 7E C360		JMP	START	BAD OPCODE, RETURN TO MON.
C173 EE 01	MIS2	LDX	1,X	GET ROUTINE ADRS FROM TABLE
C175 96 2E		LDA A	VX	GET VX
C177 6E 00		JMP	0,X	GO TO ROUTINE

C179 96 20	VTIME	LDA A	TIME	
C17B 20 B0		BRA	PUTVX	
C17D BD C2C4	VKEY	JSR	GETKEY	
C180 20 AB		BRA	PUTVX	
C182 97 20	TIMEV	STA A	TIME	
C184 39		RTS		
C185 16	TONEV	TAB		SET DURATION=VX
C186 7E C2E1		JMP	BTONE	
C189 5F	LETIV	CLR B		16-BIT ADD VX TO I
C18A 9B 27		ADD A	I+1	
C18C 97 27		STA A	I+1	
C18E D9 26		ADC B	I	
C190 D7 26		STA B	I	
C192 39		RTS		

*** COPY COMPRESSED DIGIT PATTERN (FROM TABLE)
TO 5-BYTE ARRAY (DDPAT), & SET I FOR 'SHOW'.

C193 CE C1BC	LETDSP	LDX	#HEXTAB-2	POINT TO HEX DIGIT PATTERNS
C196 84 0F		AND A	#0F	ISOLATE LS DIGIT
C198 08	LDSP1	INX		SEARCH TABLE.....
C199 08		INX		
C19A 4A		DEC A		(A=VX)
C19B 2A FB		BPL	LDSP1	
C19D EE 00		LDX	0,X	MOVE PATNH
C19F DF 1E		STX	PATNH	
C1A1 CE 0008		LDX	#DDPAT	POINT PATRN ARRAY(5)
C1A4 DF 26		STX	I	SET MI POINTER
C1A6 C6 05		LDA B	#5	DO 5 TIMES.....
C1A8 96 1E	LDSP5	LDA A	PATNH	
C1AA 84 E0		AND A	#E0	EXTRACT 3 BITS
C1AC A7 04		STA A	4,X	
C1AE 09		DEX		
C1AF 86 03		LDA A	#3	DO 3 TIMES...
C1B1 79 001F	LDSP3	ROL	PATNL	MOVE NEXT 3 BITS
C1B4 79 001E		ROL	PATNH	
C1B7 4A		DEC A		
C1B8 26 F7		BNE	LDSP3	CONT(3)....
C1BA 5A		DEC B		
C1BB 26 EB		BNE	LDSP5	CONT(5)....
C1BD 39		RTS		

*** HEXADECIMAL DIGIT PATTERNS (3*5 MATRIX):

C1BE	F6	DF	HEXTAB	FDB	\$F6DF	0
C1C0	49	25		FDB	\$4925	1
C1C2	F3	9F		FDB	\$F39F	2
C1C4	E7	9F		FDB	\$E79F	3
C1C6	3E	D9		FDB	\$3ED9	4
C1C8	E7	CF		FDB	\$E7CF	5
C1CA	F7	CF		FDB	\$F7CF	6
C1CC	24	9F		FDB	\$249F	7
C1CE	F7	DF		FDB	\$F7DF	8
C1D0	E7	DF		FDB	\$E7DF	9
C1D2	B7	DF		FDB	\$B7DF	A
C1D4	D7	DD		FDB	\$D7DD	B
C1D6	F2	4F		FDB	\$F24F	C
C1D8	D6	DD		FDB	\$D6DD	D
C1DA	F3	CF		FDB	\$F3CF	E
C1DC	93	4F		FDB	\$934F	F

C1DE	DE	26	LETDEQ	LDX	I	GET MI POINTER
C1E0	C6	64		LDA	B #100	N=100
C1E2	8D	06		BSR	DECI	CALC 100'S DIGIT
C1E4	C6	0A		LDA	B #10	N=10
C1E6	8D	02		BSR	DECI	CALC 10'S DIGIT
C1E8	C6	01		LDA	B #1	
C1EA	D7	0E	DECI	STA	B N	
C1EC	5F			CLR	B	
C1ED	91	0E	LDEQ1	CMP	A N	DO UNTIL A<N ...
C1EF	25	05		BCS	LDEQ2	BRANCH IF LOWER; NOT SAME.
C1F1	5C			INC	B	
C1F2	90	0E		SUB	A N	
C1F4	20	F7		BRA	LDEQ1	END-DO...
C1F6	E7	00	LDEQ2	STA	B 0,X	STASH
C1F8	08			INX		FOR NEXT DIGIT
C1F9	39			RTS		

C1FA	0F		STORV	SEI		KILL IRQ FOR DATA STACK
C1FB	9F	12		STS	XTEMP	SAVE SP
C1FD	8E	002F		LDS	#V0-1	POINT TO VARIABLES TABLE
C200	DE	26		LDX	I	POINT MI
C202	20	09		BRA	MOVX	TRANSFER NB BYTES
C204	0F		LOADV	SEI		KILL IRQ
C205	9F	12		STS	XTEMP	
C207	9E	26		LDS	I	POINT MI
C209	34			DES		
C20A	CE	0030		LDX	#V0	POINT TO V0
C20D	D6	2B	MOVX	LDA	B VXLOC+1	CALC. X (AS IN VX)
C20F	C4	0F		AND	B #\$0F	LOOP (X+1) TIMES.....
C211	32		MOVX1	PUL	A	GET NEXT V
C212	A7	00		STA	A 0,X	COPY IT
C214	08			INX		
C215	7C	0027		INC	I+1	I=I+X+1 (ASSUMES SAME PAGE)
C218	5A			DEC	B	
C219	2A	F6		BPL	MOVX1	CONTINUE....
C21B	9E	12		LDS	XTEMP	RESTORE SP
C21D	0E			CLI		RESTORE IRQ
C21E	39			RTS		

*** DISPLAY ROUTINES *****

C21F D6 29	SHOW	LDA B	PIR+1	GET N (OPCODE LSB)
C221 7F 003F		CLR	VF	CLEAR OVERLAP FLAG
C224 DE 26	SHOWI	LDX	I	POINT TO PATTERN BYTES
C226 86 01	SHOWX	LDA A	#\$01	** SET DISPLAY ADRS MSB =
C228 97 1C		STA A	BLOC	= DISBUF HIGH-ORDER BYTE.
C22A C4 0F		AND B	#\$0F	COMPUTE NO. OF BYTES (N)
C22C 26 02		BNE	SHOW2	IF N=0, MAKE N=16
C22E C6 10		LDA B	#16	
C230 37	SHOW2	PSH B		DO N TIMES.....
C231 DF 14		STX	ZHI	SAVE MI POINTER
C233 A6 00		LDA A	0,X	FETCH NEW PATTERN BYTE
C235 97 1E		STA A	PATNH	
C237 7F 001F		CLR	PATNL	
C23A D6 2E		LDA B	VX	DETERMINE OFFSET BIT COUNT
C23C C4 07		AND B	#7	
C23E 27 09	SHOW3	BEQ	SHOW4	SHIFT INTO MATCHING POS'N
C240 74 001E		LSR	PATNH	
C243 76 001F		ROR	PATNL	
C246 5A		DEC B		
C247 26 F5		BNE	SHOW3	
C249 D6 2E	SHOW4	LDA B	VX	GET X COORD
C24B 8D 28		BSR	DISLOC	FIND WHERE IS FIRST DISP BYT
C24D 96 1E		LDA A	PATNH	
C24F 8D 15		BSR	SHOWUP	
C251 D6 2E		LDA B	VX	
C253 CB 08		ADD B	#8	FIND WHERE IS ADJACENT BYTE
C255 8D 1E		BSR	DISLOC	
C257 96 1F		LDA A	PATNL	
C259 8D 0B		BSR	SHOWUP	
C25B 7C 002F		INC	VY	
C25E DE 14		LDX	ZHI	POINT NEXT PATTERN BYTE
C260 08		INX		
C261 33		PUL B		
C262 5A		DEC B		
C263 26 CB		BNE	SHOW2	CONT.....
C265 39		RTS		
C266 16	SHOWUP	TAB		UPDATE DISPLAY BYTE
C267 E8 00		EOR B	0,X	X-OR WITH EXISTING DISPLAY
C269 AA 00		ORA A	0,X	'OR' ALSO, FOR O'LAP TEST
C26B E7 00		STA B	0,X	STORE XOR'ED BYTE
C26D 11		CBA		
C26E 27 04		BEQ	SHOWR	'XOR' SAME AS 'OR', ELSE...
C270 86 01		LDA A	#1	SET OVERLAP FLAG (VF)
C272 97 3F		STA A	VF	
C274 39	SHOWR	RTS		


```

*** COMPUTE ADRS OF DISPLAY BYTE AT COORDS (B,VY):
C275 96 2F  DISLOC LDA A  VY      FETCH Y COORD
C277 84 1F      AND A  #$1F      MASK TO 5 BITS FOR WRAP-ROUND
C279 48          ASL A          LEFT JUSTIFY
C27A 48          ASL A
C27B 48          ASL A
C27C C4 3F      AND B  #$3F      MASK X COORD TO 6 BITS
C27E 54          LSR B          DROP 3 LS BITS
C27F 54          LSR B
C280 54          LSR B
C281 1B          ABA
C282 97 1D      STA A  BLOC+1     BUILD BYTE
C284 DE 1C      LDX  BLOC         DISP LOC'N LSB COMPLETED
C286 39          RTS             POINT TO DISP BYTE AT (VX,VY)

```

```

*** KEYPAD ROUTINES *****
C287 C6 F0  PAINZ LDA B  #$F0     INITIALIZE PORT
C289 CE 8010 PAINV LDX  #PIAA     (ENTRY PT FOR INV. DDR)
C28C 6F 01      CLR   1,X        RESET & SELECT DDR
C28E E7 00      STA B  0,X        SET DATA DIRECTION
C290 C6 06      LDA B  #$06      SEL O/P REG & SETUP CTRL
C292 E7 01      STA B  1,X
C294 6F 00      CLR   0,X        OUTPUT ZEROS & RESET FLAGS
C296 39          RTS

```

```

*** KEYPAD INPUT SERVICE ROUTINE *****
C297 8D EE  KEYINP BSR  PAINZ     RESET KEYPAD PORT
C299 7F 0018  CLR   BADRED      RESET BAD-READ FLAG
C29C 8D 55  BSR   DEL333      DELAY FOR DEBOUNCE
C29E E6 00  LDA B  0,X        INPUT ROW DATA
C2A0 8D 15  BSR   KBILD       FORM CODE BITS 0,1
C2A2 97 17  STA A  KEYCOD
C2A4 C6 0F  LDA B  #$0F       SET DDR FOR....
C2A6 8D E1  BSR   PAINV      INVERSE ROW/COL DIR'N
C2A8 E6 00  LDA B  0,X        INPUT COLUMN DATA
C2AA 54      LSR B          RIGHT JUSTIFY
C2AB 54      LSR B
C2AC 54      LSR B
C2AD 54      LSR B
C2AE 8D 07  BSR   KBILD       FORM CODE BITS 2,3
C2B0 48      ASL A
C2B1 48      ASL A
C2B2 9B 17  ADD A  KEYCOD
C2B4 97 17  STA A  KEYCOD     BUILD COMPLETE KEY CODE
C2B6 39      RTS

C2B7 C1 0F  KBILD CMP B  #$0F     CHECK KEY STATUS
C2B9 26 02  BNE   KBILDO      KEY IS DOWN, GO DECODE IT
C2BB D7 18  STA B  BADRED     NO KEY, SET BAD-READ FLAG
C2BD 86 FF  KBILDO LDA A  #-1    (A=RESULT)
C2BF 4C      KBILD1 INC A      SHIFT DATA BIT TO CARRY
C2C0 54      LSR B          FOUND ZERO BIT ?
C2C1 25 FC  BCS   KBILD1
C2C3 39      RTS

```



```

*** GETKEY WAITS FOR KEYDOWN, THEN INPUTS *****
C2C4 DF 12   GETKEY STX   XTEMP   SAVE X FOR CALLING ROUTINE
C2C6 8D BF   GETK1  BSR    PAINZ   RESET PORT, CLEAR FLAGS
C2C8 A6 01   GETK2  LDA A   1,X    INPUT STATUS (HEX KEY DOWN?)
C2CA 2B 07           BMI    HEXKEY  YES; FETCH IT IN
C2CC 48           ASL A           TRY CA2 FLAG
C2CD 2A F9           BPL    GETK2   FN KEY DOWN ? (AC0?)
C2CF 6D 00   FNKEY  TST    0,X    YES; RESET FLAG IN PIA
C2D1 20 07           BRA    HEXK1   RETURN WITHOUT CODE
C2D3 8D C2   HEXKEY BSR    KEYINP  DECODE THE KEYPAD
C2D5 7D 0018  TST    BADRED  WAS IT A BAD READ ?
C2D8 26 EC           BNE    GETK1   YES, TRY AGAIN
C2DA 8D 03   HEXK1  BSR    BLEEP   O.K. ACKNOWLEDGE
C2DC DE 12           LDX    XTEMP  RESTORE CALLER'S X-REG
C2DE 39           RTS           RETURN (WITH AC0 FOR FN KEY)

```

```

*** TONE GENERATING ROUTINES
C2DF C6 04   BLEEP  LDA B #4
C2E1 D7 21   BTONE  STA B  TONE   SET DURATION (RTC CYCLES)
C2E3 C6 41           LDA B #$41   TURN AUDIO ON
C2E5 F7 8012           STA B  PIAB
C2E8 7D 0021  BTON1  TST    TONE   WAIT FOR RTC TIME-OUT
C2EB 26 FB           BNE    BTON1
C2ED C6 01           LDA B #1     TURN AUDIO OFF
C2EF F7 8012           STA B  PIAB
C2F2 39           RTS

```

```

*** SOFTWARE DELAY ROUTINE FOR SERIAL I/O:
C2F3 8D 00   DEL333 BSR    DEL167  DELAY FOR 3.33 MILLISEC
C2F5 37           DEL167 PSH B
C2F6 C6 C8           LDA B #200    DELAY FOR 1.67 MILLISEC
C2F8 5A   DEL    DEC B
C2F9 01           NOP
C2FA 26 FC           BNE    DEL
C2FC 33           PUL B
C2FD 39           RTS

```

*** TAPE INPUT/OUTPUT ROUTINES *****

```

*** INITIALIZE TAPE, TONE, RTC, & DMA ***
*** A=$3F FOR DISPLAY/DMA ON; A=$37 FOR OFF:
C2FE CE 8012  PBINZ  LDX    #PIAB
C301 C6 3B           LDA B #$3B   SELECT DDR (DMA ON)
C303 E7 01           STA B  1,X
C305 C6 7F           LDA B #$7F   WRITE DDR
C307 E7 00           STA B  0,X
C309 A7 01           STA A  1,X   WRITE CTRL REG
C30B C6 01           LDA B #1     OUTPUT FOR TONE OFF, AND...
C30D E7 00           STA B  0,X   TAPE DATA-OUT HIGH (MARKING)
C30F 39           RTS

```



```

*** INPUT ONE BYTE FROM TAPE PORT ***
C310 8D 13  INBYT  BSR  XCHG  EXCHANGE X FOR PIA ADRS
C312 A6 00  IN1    LDA  A  0,X
C314 2B FC   BMI    IN1    LOOK FOR START BIT
C316 8D DD   BSR    DEL167  DELAY HALF BIT-TIME (300BD)
C318 C6 09   LDA  B  #9    DO 9 TIMES....
C31A 0D     IN2    SEC     ENSURE PBO MARKING
C31B 69 00   ROL    0,X    INPUT & SHIFT NEXT BIT
C31D 46     ROR  A    INTO ACC-A
C31E 8D D3   BSR    DEL333  WAIT 1 BIT-TIME
C320 5A     DEC  B
C321 26 F7   BNE    IN2    CONT.....
C323 20 17   BRA    OUTX   RESTORE X AND RETURN

C325 DF 12   XCHG   STX    XTEMP  SAVE X-REG
C327 CE 8012 LDX    #PIAB
C32A 39     RTS

```

```

*** OUTPUT ONE BYTE TO TAPE PORT ***
C32B 8D F8  OUTBYT BSR  XCHG
C32D 36     PSH  A
C32E 6A 00   DEC    0,X    RESET START BIT
C330 C6 0A   LDA  B  #10   DO 10 TIMES.....
C332 8D BF   OUT1  BSR  DEL333 DELAY 1 BIT-TIME
C334 A7 00   STA  A  0,X    NEXT BIT TO OUT LINE (PBO)
C336 0D     SEC
C337 46     ROR  A
C338 5A     DEC  B
C339 26 F7   BNE    OUT1   CONT.....
C33B 32     PUL  A    RESTORE A.
C33C DE 12   OUTX  LDX    XTEMP  RESTORE X.
C33E 39     RTS

```

```

C33F 20 83  GETKEE BRA  GETKEY  FOR INTERLINKING

```

```

*** TAPE LOAD AND DUMP ROUTINES *****
C341 86 37  LODUMX LDA  A  #37  KILL DISPLAY (DMA OFF)
C343 8D B9   BSR    PBINZ
C345 DE 02   LDX    BEGA    POINT TO FIRST LOAD/DUMP ADR
C347 39     RTS
C348 8D F7   DUMP  BSR    LODUMX
C34A A6 00   DUMP1 LDA  A  0,X  FETCH RAM BYTE
C34C 8D DD   BSR    OUTBYT
C34E 08     INX
C34F 9C 04   CPX    ENDA    (END = LAST ADRS + 1)
C351 26 F7   BNE    DUMP1
C353 20 0B   BRA    START
C355 8D EA   LOAD  BSR    LODUMX
C357 8D B7   LOAD1 BSR    INBYT
C359 A7 00   STA  A  0,X    STASH BYTE IN RAM
C35B 08     INX
C35C 9C 04   CPX    ENDA    DONE?
C35E 26 F7   BNE    LOAD1   CONT....
*** (BRA  START) ***

```

***** MONITOR ENTRY POINT *****

```

C360 8E 007F  START  LDS  #STOP  RESET SP TO TOP
C363 CE C3E9      LDX  #RTC    SETUP IRQ VECTOR FOR RTC
C366 DF 00        STX   IRQV
C368 86 3F        LDA  A  #$3F  SETUP I/O PORT; DISPLAY ON.
C36A 8D 92        BSR   PBINZ
C36C 8D 43        BSR   SHOADR  PROMPT
C36E 0E           CLI
C36F 8D CE        BSR   GETKEE  ENABLE RELATIVE TIME CLOCK
                                INPUT SOMETHING
C371 4D           TST  A
C372 2A 10        BPL   INADRS  IF HEX, GET AN ADDRESS
C374 8D C9        BSR   GETKEE  IF FN, GET A COMMAND
C376 84 03        AND  A  #3
C378 27 23        BEQ   MEMOD   0 = MEMORY MODIFY
C37A 4A           DEC  A
C37B 27 D8        BEQ   LOAD    1 = TAPE LOAD
C37D 4A           DEC  A
C37E 27 C8        BEQ   DUMP    2 = TAPE DUMP
C380 DE 06        GO    LDX  ADRS  FETCH ADRS FOR GO
C382 6E 00        JMP   0,X
                                0,X = 0

C384 8D 0C        INADRS BSR   BYT1  BUILD ADRS MS BYTE
C386 97 06        STA  A  ADRS
C388 8D 06        BSR   BYTIN  INPUT & BUILD LSB
C38A 97 07        STA  A  ADRS+1
C38C 8D 23        BSR   SHOADR  DISPLAY RESULTANT ADRS
C38E 20 DF        BRA   COMAND

C390 8D AD        BYTIN  BSR   GETKEE  INPUT 2 HEX DIGITS
C392 48           BYT1  ASL  A  LEFT JUSTIFY FIRST DIGIT
C393 48           ASL  A
C394 48           ASL  A
C395 48           ASL  A
C396 97 0F        STA  A  ATEMP  HOLD IT
C398 8D A5        BSR   GETKEE  INPUT ANOTHER DIGIT
C39A 9B 0F        ADD  A  ATEMP  BUILD A BYTE
C39C 39           RTS

```

*** MEMORY MODIFY ROUTINE

```

C39D 8D 12        MEMOD BSR   SHOADR  SHOW CURRENT ADRS
C39F DE 06        LDX   ADRS    SHOW DATA AT ADRS
C3A1 8D 25        BSR   SHODAT
C3A3 8D 9A        BSR   GETKEE  WAIT FOR INPUT
C3A5 4D           TST  A
C3A6 2B 04        BMI   MEM1    FN KEY; NEXT ADRS
C3A8 8D E8        BSR   BYT1    HEX KEY; NEW DATA BYTE
C3AA A7 00        STA  A  0,X   DEPOSIT IT
C3AC 08           MEM1  INX
C3AD DF 06        STX   ADRS    BUMP ADRS
C3AF 20 EC        BRA   MEMOD

```

C3B1	86	10	SHOADR	LDA	A	##10	SET CURSOR HOME POSN
C3B3	8D	2B		BSR		CURS1	
C3B5	CE	01C8		LDX		#DISBUF+200	ILLUMINATE LAST 7 ROWS
C3B8	86	FF		LDA	A	##FF	
C3BA	BD	C07D		JSR		FILL	
C3BD	CE	0006		LDX		#ADRS	POINT TO ADRS MS BYTE
C3C0	8D	06		BSR		SHODAT	
C3C2	08			INX			POINT TO ADRS LS BYTE
C3C3	8D	03		BSR		SHODAT	
C3C5	8D	15		BSR		CURSR	MOVE CURSOR RIGHT
C3C7	39			RTS			
C3C8	A6	00	SHODAT	LDA	A	0,X	FETCH DATA @ X
C3CA	36		SHOBYT	PSH	A		
C3CB	44			LSR	A		ISOLATE MS DIGIT
C3CC	44			LSR	A		
C3CD	44			LSR	A		
C3CE	44			LSR	A		
C3CF	8D	01		BSR		DIGOUT	SHOW ONE DIGIT
C3D1	32			PUL	A		
C3D2	DF	12	DIGOUT	STX		XTEMP	SAVE X
C3D4	BD	C193		JSR		LETDSP	POINT TO DIGIT PATTERN
C3D7	C6	05		LDA	B	#5	SHOW 5-BYTE PATTERN
C3D9	BD	C224		JSR		SHOWI	
C3DC	86	04	CURSR	LDA	A	#4	SHIFT CURSOR RIGHT 4 DOTS
C3DE	9B	2E		ADD	A	VX	
C3E0	97	2E	CURS1	STA	A	VX	SET X COORD
C3E2	86	1A		LDA	A	##1A	SET Y COORD
C3E4	97	2F		STA	A	VY	
C3E6	DE	12		LDX		XTEMP	RESTORE X-REG
C3E8	39			RTS			

*** REAL TIME CLOCK INTERRUPT SERVICE ROUTINE:

C3E9	7A	0020	RTC	DEC		TIME	
C3EC	7A	0021		DEC		TIME	
C3EF	7D	8012		TST		PIAB	CLEAR IRQ FLAG IN PIA
C3F2	3B			RTI			

C3F3	DE	00	IRQ	LDX		IRQV	INDIRECT JUMP VIA IRQV
C3F5	6E	00		JMP		0,X	
C3F7	00			FCB		0	

*** RESTART AND INTERRUPT TRAPS:

C3F8	C3	F3	FDB		IRQ	(ALLS USER-WRITTEN ISR)
C3FA	00	80	FDB		\$0080	SWI ROUTINE AT \$0080 (OPTION
C3FC	00	83	FDB		\$0083	NMI ROUTINE AT \$0083 (OPTION
C3FE	C3	60	FDB		START	

END CHIPOS INTERPRETER/MONITOR


```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
M6800 MICROPROCESSOR CROSS-ASSEMBLER
[ASM68 REV.II (FORTRAN) BY M.J. BAUER, 1978]
DEAKIN UNIVERSITY DEC-SYSTEM-20/50
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

* DREAMBUG *

```

```

*
* DE-BUGGING UTILITY ROUTINE
* FOR MACHINE-CODE PROGRAMS *

```

```

* M.J. BAUER, APR'79.

```

```

*****

```

```

* A 'SWI' INSTRUCTION IN THE PROGRAM UNDER TEST
* WILL CAUSE A JUMP TO 'DREAMBUG', WHICH THEN
* DISPLAYS THE REGISTERS, THUS:-

```

```

* CC B A X PC

```

```

* WHEN ANY HEX KEY IS PRESSED, EXECUTION OF THE
* MAIN PROGRAM WILL BE RESUMED.
* TO EXAMINE THE STACK-PTR (AS IT WAS WHEN THE
* BREAK OCCURED); RESET, THEN EXAMINE LOC'N $0010.

```

```

* N.B: BREAKPOINTS ARE INSERTED INTO THE PROGRAM
* BY REPLACING THE LAST BYTE OF ANY CONVENIENT
* INSTRUCTION WITH A 'SWI' [3F]. PRECEDING BYTES
* OF 2 AND 3 BYTE INSTRN'S ARE REPLACED WITH 'NOP'S
* [01]; THIS ENSURES CORRECT CONTINUATION.

```

```

*****

```

```

* EXTERNAL (CHIPOS) REFERENCES *

```

```

0006      ADRES      EQU      $0006
0010      SP         EQU      $0010      STACK-POINTER SAVE
0016      VSTAT      EQU      $0016      I/O STATUS SAVE
C3C8      SHODAT     EQU      $C3C8
C3DC      CURSR      EQU      $C3DC
C07D      FILL       EQU      $C07D
C287      PAINZ      EQU      $C287
C2F3      DEL333     EQU      $C2F3

```

```

* SOFTWARE INTERRUPT ENTRY POINT *

```

```

** [ RELOCATABLE BY CHANGING JMP ADRES AT 0080 ]

```

```

0080      ORG        $0080

```

```

0080 7E 0083  SWI      JMP      DBUG
0083 B6 8013  DBUG     LDA A    $8013      SAVE PORT-B STATUS
0086 97 16    STA A    VSTAT
0088 8A 08    ORA A    #$08      TURN VIDEO ON
008A B7 8013  STA A    $8013
008D 86 02    LDA A    #2      CURSOR HOME
008F BD C3E0  JSR      CURSR+4

```



```

0092 CE 01C8      LDX  #$01C8      WHITE OUT DISPLAY WINDOW
0095 86 FF        LDA  A  #$FF
0097 BD C07D      JSR   FILL

*
*  DISPLAY REGISTERS ON STACK
*

009A 30          TSX
009B C6 04        LDA  B  #4      POINT TO REG. IMAGES ON STK
                                DO 4 TIMES.....
009D 37          DBG4  PSH  B
009E BD C3C8      JSR   SHODAT    SHOW DATA BYTE AT X
00A1 86 01        LDA  A  #1      FWD SPACE
00A3 BD C3DE      JSR   CURSR+2
00A6 08          INX
00A7 33          PUL  B
00A8 5A          DEC  B
00A9 26 F2        BNE   DBG4      CONTINUE.....
00AB 86 FF        LDA  A  #-1     BACK SPACE
00AD BD C3DE      JSR   CURSR+2
00B0 08          INX      SHOW X (LSB)
00B1 BD C3C8      JSR   SHODAT
00B4 86 02        LDA  A  #2      FWD SPACE
00B6 BD C3DE      JSR   CURSR+2
00B9 08          INX      SHOW PC (MSB)
00BA BD C3C8      JSR   SHODAT
00BD 08          INX      SHOW PC (LSB)
00BE BD C3C8      JSR   SHODAT
00C1 DF 10        STX   SP        SAVE USER'S SP
00C3 BD C287      JSR   PAINZ     WAIT FOR KEYDOWN
00C6 7D 8011      WAIT  TST  $8011
00C9 2A FB        BPL   WAIT
00CB BD C2F3      JSR   DEL333    DE-BOUNCE DELAY
00CE BD C2F3      JSR   DEL333
00D1 CE 01C8      LDX  #$01C8    ERASE WINDOW
00D4 4F          CLR  A
00D5 BD C07D      JSR   FILL
00D8 96 16        LDA  A  VSTAT   RESTORE I/O STATUS
00DA B7 8013      STA  A  $8013
00DD 3B          RTI      CONTINUE USER PROGRAM

*
*  END 'DREAMBUG' UTILITY

```

[illegible]

M6800 MICROPROCESSOR CROSS-ASSEMBLER
 [ASM68 REV.II (FORTRAN) BY M.J. BAUER, 1978]
 DEAKIN UNIVERSITY DEC-SYSTEM-20/50

```

*
*  MEMORY CHECK PROGRAM FOR D.R.E.A.M.6800
*
*****
*
*  THIS PROGRAM REQUIRES 'DREAMBUG' TO BE LOADED.
*  TO USE, FIRST PUT THE STARTING AND ENDING LOCATION
*  OF THE BLOCK TO BE CHECKED AT 0002-0005, AS FOR
*  TAPE LOAD/DUMP. THEN GO FROM 0200. IF A RAM
*  ERROR IS FOUND, CONTROL WILL BE PASSED TO DREAMBUG
*
*****

```

0002	START	EQU	\$0002	BEGIN ADRS FOR CHECK
0004	ENDLOC	EQU	\$0004	END ADRS FOR CHECK
0019	FLAG	EQU	\$0019	
001A	XTEMP	EQU	\$001A	
0200		ORG	\$0200	
0200 7F 0019	GO	CLR	FLAG	FLAG=0 ON FIRST SCAN
0203 DE 02	OVER	LDX	START	POINT TO START LOC'N
0205 86 FF	NEXT	LDA A	#\$FF	
0207 A7 00		STA A	0,X	WRITE ALL ONES
0209 E6 00		LDA B	0,X	READ BACK
020B 11		CBA		COMPARE (CHECK ONES)
020C 27 01		BEQ	GREEN	O.K. CONTINUE
020E 3F		SWI		DATA ERROR, ABORT TEST
020F 7D 0019	GREEN	TST	FLAG	ON FIRST SCAN?
0212 27 0E		BEQ	NEW	YES, FORGET ZEROS
0214 DF 1A		STX	XTEMP	CHECK ZEROS, SAVE X
0216 08	LOOP	INX		BUMP POINTER
0217 9C 04		CPX	ENDLOC	FINISHED?
0219 27 05		BEQ	RESTOR	DONE, CONT.
021B E6 00		LDA B	0,X	CHECK FOR ZERO STORED
021D 27 F7		BEQ	LOOP	ZERO, GOOD
021F 3F		SWI		ADRS ERROR OR BIT STUCK HIGH
0220 DE 1A	RESTOR	LDX	XTEMP	GET POINTER
0222 6F 00	NEW	CLR	0,X	WRITE ZEROS
0224 08		INX		BUMP POINTER
0225 9C 04		CPX	ENDLOC	DONE?
0227 26 DC		BNE	NEXT	NO, NEXT LOC
0229 7C 0019		INC	FLAG	ADD 1 TO SCAN COUNT
022C 20 D5		BRA	OVER	
	END		MEMORY CHECK	


```

1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

```

M6800 MICROPROCESSOR CROSS-ASSEMBLER
 [ASM68 REV.II (FORTRAN) BY M.J. BAUER, 1978]
 DEAKIN UNIVERSITY DEC-SYSTEM-20/50

```

%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

* BRANCH OFFSET CALCULATOR

* INSTRUCTIONS FOR USE:

* NOTE THE ADRS TO BE BRANCHED TO. ALSO NOTE
 * THE ADRS OF THE NEXT INSTRN AFTER THE BRANCH
 * INSTRN. THEN RUN THE PRGM (GO FROM 0200).
 * ENTER THE LOWER OF THE TWO ADDRESSES (4 HEX),
 * THEN THE HIGHER ONE. THE COMPUTER SHOWS: 2
 * RESULTS: THE FIRST FOR FORWARD BRANCHING,
 * SECOND FOR BRANCHING BACK. IF THE OPERAND IS
 * OUT OF RANGE, A PECULIAR SYMBOL WILL BE SHOWN.
 * PRESS ANY KEY TO RERUN THE PROGRAM.

* EXTERNAL REFS:

001A	LOC1	EQU	\$001A	2-BYTE TEMP
C079	ERASE	EQU	\$C079	
C3E0	CURS1	EQU	\$C3E0	
C226	SHOWX	EQU	\$C226	
C2C4	GETKEY	EQU	\$C2C4	
C3CA	SHOBYT	EQU	\$C3CA	
C3DC	CURSR	EQU	\$C3DC	
C390	BYTIN	EQU	\$C390	
0006	ADRS	EQU	\$0006	
C3B1	SHOADR	EQU	\$C3B1	

*

*


```

0200          ORG      $0200
          * [ RELOCATABLE WITHOUT MODIFICATION ]
          *
0200 BD C079  GO      JSR      ERASE
0203 86 02          LDA A #2
0205 BD C3E0          JSR      CURS1
0208 8D 2E      IN    BSR      INUM      INPUT & SHOW 4-DIGIT ADRS
020A DF 1A          STX      LOC1      SAVE IT
020C 8D 2A          BSR      INUM      INPUT & SHOW 2ND ADRS
020E 4F          CLR A
020F C6 80          LDA B #$80      DO 128 TIMES (MAX).....
0211 9C 1A      LOOP  CPX      LOC1      DONE?
0213 27 12          BEQ      DSPANS     YES, DISPLAY ANSWER.
0215 4C          INC A
0216 09          DEX
0217 5A          DEC B
0218 26 F7          BNE      LOOP      CONTINUE.....
021A CE C09F          LDX      #$C09F     SHOW GARBAGE (ERROR)
021D C6 05          LDA B #5
021F BD C226          JSR      SHOWX
0222 BD C2C4          JSR      GETKEY     WAIT
0225 20 D9          BRA      GO          TRY AGAIN

          *
0227 36      DSPANS  PSH A
0228 BD C3CA          JSR      SHOBYT     SHOW +++ RESULT
022B BD C3DC          JSR      CURSR
022E 32          PUL A
022F 40          NEG A
0230 BD C3CA          JSR      SHOBYT     SHOW --- RESULT
0233 BD C2C4          JSR      GETKEY     WAIT
0236 20 C8          BRA      GO          REPEAT....

          *
0238 BD C390      INUM  JSR      BYTIN     INPUT 2 HEX
023B 97 06          STA A  ADRS          STASH
023D BD C390          JSR      BYTIN
0240 97 07          STA A  ADRS+1
0242 BD C3BD          JSR      SHOADR+12  SHOW 4 HEX DIGITS
0245 DE 06          LDX      ADRS          FETCH SAME
0247 39          RTS

          *
          END      BRANCH OFFSET CALCULATOR

```

1
 //////////////////////////////////////
 SYMBOL TABLE
 //////////////////////////////////////

001A	LOC1	C2C4	GETKEY	0006	ADRS	0211	LOOP
C079	ERASE	C3CA	SHOBYT	C3B1	SHOADR	0227	DSPANS
C3E0	CURS1	C3DC	CURSR	0200	GO	0238	INUM
C226	SHOWX	C390	BYTIN	0208	IN	0000	

////////////////////////////////////

GETTING STARTED WITH CHIP-8

CHIP-8 is a hexadecimal interpretive language with an instruction set geared toward games, graphics, and keyboard sampling to manipulate a video display.

A hexadecimal interpreter provides great memory savings in that no alphanumeric character strings need to be saved; only compact codes containing the operation to be performed and the parameters.

All CHIP-8 instructions are contained in 2 bytes (or 4 hex digits). This makes changing code or the insertion of debugging instructions (i.e., a program stop) easier. This uniform size also helps to eliminate addressing errors.

The hexadecimal format and the 16 memory registers (variables) of CHIP-8 create a pseudo machine language which is very powerful for games and graphics applications. The 16 variables are used very much like machine registers but are easier to access. Most of the language's instructions refer to or manipulate one or more of these variables.

The video display is implemented by mapping the contents of the second page (256 bytes) of on-board memory to the display screen. The video interface on the bus generates the timing for the display including the generation of the display refresh interrupt request.

The function of CHIP-8 programs, therefore, is just to arrange the data in the display page to provide the desired graphics.

The first byte of the display page will be shown in the upper left of the display screen with successive bytes following from left to right in 32 rows of 8 bytes each. Each row of 8 bytes is actually shown on four successive scan lines to give each bit a vertical height nearly equal to its horizontal width. This vertical height is set by the display refresh logic. Writing to the display page involves specifying the position the data should occupy in the display and then transferring a list of bytes to that position.

The display page is made to look like a coordinate grid by CHIP-8. Since the display consists of a 64x32 bit format (40x20 in hex values) any bit position can be specified by a coordinate pair. The horizontal coordinates range from 00 at the left to 3F at the right of the screen. The vertical coordinates range from 00 at the top to 1F at the bottom of the screen.

Before transferring a pattern to the screen, a memory pointer must be set to the first byte of the pattern list. This will be explained in an example later. When the CHIP-8 display instruction transfers the pattern list to the display memory the most significant bit of the first byte in the list is posi-

tioned at the specified bit coordinate. This may require the pattern byte to be shifted several bits to the right and be shown in two horizontally adjacent bytes. CHIP-8 takes care of this shifting and also insures that all the bytes in the pattern list line up vertically.

There are 31 CHIP-8 instructions. They will be broken down into 8 groups: Set Variable, Branch and Call, Conditional Branch, Random Number Generator, Keyboard, Arithmetic and Logic, Tone and Timer, Memory Pointer, and Display. In the following descriptions, X and Y refer to variable names and KK is any hexadecimal constant. Use of variable 0 (V0) is mandatory for certain instructions, and variable F (VF) is altered during the use of certain instructions.

Set Variable Instructions

- 6XKK Set the variable indicated by X to KK. Any variable may be set to any value 00 through FF by this instruction. For instance, an instruction 6304 sets V3 to a value of 04.
- 7XKK Increment the variable indicated by X by KK. Any variable may be incremented by any amount from 00 to FF. If the result is greater than FF the overflow is not saved but the value saved in variable X will be correct. Incrementing by FF appears to be a decrement by 01, FE decrements by 2, etc.
- 8XY0 Copy the current value of the variable indicated by Y into the variable indicated by X. The value of any variable may be copied into any other variable.

Branch and Call Instructions

- 1MMM Jump to the CHIP-8 instruction at memory location OMMM. The Ms are replaced by 3 hexadecimal digits specifying the address of some CHIP-8 instruction which is to be the next instruction executed. The range of addresses possible is from 0000 to 0FFF. Addresses over 0FFF are not accessible due to the two byte CHIP-8 format. Branching to locations under 0200 is discouraged because these locations contain the CHIP-8 video buffer and scratchpad.
- BMMM Add the value of variable V0 to OMMM and jump to the CHIP-8 instruction at the resulting memory location. If the result of adding V0 to OMMM increments the low order address past FF then the high order address is incremented. This instruction may be used to branch to one of several locations based on the results of previous events.

2MMM Calls a CHIP-8 subroutine at memory location OMMM. May be used to call a CHIP-8 instruction sequence from more than one place in the program and, when finished, return to the instruction immediately following the call.

Subroutines may also be called from within a subroutine. This is known as nesting. In CHIP-8, nesting may be made up to 12 levels deep, before conflicting with the CHIP-8 memory map.

00EE Terminate a CHIP-8 subroutine. Returns execution to the CHIP-8 instruction immediately following the source of a CHIP-8 subroutine call. Every CHIP-8 subroutine must end with an 00EE instruction. Every level in subroutine nesting must end with 00EE to insure proper operation.

0MMM Calls machine language subroutine at memory location OMMM. May be used to branch execution to **M 6800** machine code to perform operations not available through CHIP-8. If Control is to be returned to CHIP-8 at the instruction immediately following the source of the call, a **39** code instruction must be used.

CXKK Probability Branching (uses the CHIP-8 Random Number Generator)

Conditional Branch Instructions

3XKK Skip the next instruction if the variable indicated by X is equal to KK. Any variable may be compared to any hex value 00 through FF; if they are equal, the next CHIP-8 instruction is skipped.

4XKK Skip the next instruction if the variable indicated by X is not equal to KK. Any variable may be compared to any hex value 00 through FF; if they are not equal the next CHIP-8 instruction is skipped.

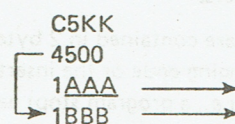
5XY0 Skip the next instruction if the variable indicated by X is equal to the variable indicated by Y. Any two variables may be compared; if they are equal the next CHIP-8 instruction is skipped.

9XY0 Skip the next instruction if the variable indicated by X is not equal to the variable indicated by Y. Any two variables may be compared; if they are not equal the next CHIP-8 instruction is skipped.

Random Number Generator

CXKK Generate a random number, logically AND it with KK, and set the variable indicated by X equal to the result. Any variable may be set to some random value. The KK is a mask and can be used to limit the range of random numbers possible. The range can be used to provide a certain number of alter-

natives of which the resulting random number selects one. The range can also be used to set the odds in a probability branching situation. If KK = 03 there is a 25% chance of the resulting random number equalling zero. An appropriate branching structure following the random number generation completes the operation. In the following example the random number is set into variable 5. The third instruction is skipped if V5 = 00. As the range of outcomes is increased the probability of the third instruction being executed decreases.



KK	%A	%B	BIT MASK possibilities
01	50	50	00000001 0 or 1
03	25	75	00000011 0,1,2 or 3
07	12.5	87.5	00000111 0 to 7
0F	6.25	93.75	00001111 0 to F
FF	.392	99.608	11111111 0 to FF

Keyboard Instructions

FX0A Wait for any key to be pressed and load the hex value into the variable indicated by X. This instruction may be used to wait for an input of data. No further instruction will be executed until some key is pressed. The next instruction will not be executed until the key is released. A tone will be generated while the key is depressed.

EX9E Skip the next instruction if the variable indicated by X is equal to key pressed on hex keyboard. This may be used to check whether a specific key on the hex keyboard has been pressed. The instruction causes the low order hex digit of the variable indicated to be compared with the keyboard. If the corresponding key is pressed the instruction will detect it, and the next instruction is skipped. If the key isn't pressed, the program doesn't wait but executes the next CHIP-8 instruction immediately.

In the following sequence, if key A is pressed V5 will not be incremented:

630A
E39E
7501

EXA1 Skip the next instruction if the variable indicated by X is not equal to the key pressed on the hex keyboard. This may be used to check whether a specific key on the hex keyboard has been pressed. The instruction causes the low order hex digit of the variable to be compared with the keyboard. If the corresponding key is pressed, the instruction will detect it, and the next instruction is executed. If the key is not pressed, the next instruction is skipped.

In the following sequence, if key A was pressed V5 would be incremented:

```

      .
      .
      .
    630A
    E3A1
    7501
      .
      .
      .
  
```

Arithmetic and Logic Instructions

8XY1 Logically OR the contents of the variable indicated by X with the contents of the variable indicated by Y and set the variable indicated by X equal to the result. Any two variables may be logically ORed together and the result stored in VX. VF is used for temporary storage and will be altered.

8XY2 Logically AND the contents of the variable indicated by X with the contents of the variable indicated by Y and set the variable indicated by X equal to the result. Any two variables may be logically ANDed together and the result stored in VX. VF will be altered.

8XY4 Add the contents of the variable indicated by Y to the contents of the variable indicated by X and store the sum in the variable indicated by X. If the sum is not more than FF, VF will be set to 00. If the sum is greater than FF, VF will be set to 01. Any two variables may be added together.

8XY5 Subtract the contents of the variable indicated by Y from the contents of the variable indicated by X and store the difference in the variable indicated by X. If the variable indicated by X was not less than the variable indicated by Y, VF will be set to 01.

If the variable indicated by X was less than the variable indicated by Y, VF will be set to 00.

Tone and Timer Instructions

FX18 Load the tone timer with the contents of the variable indicated by X. The timer is decremented 50 times per second (the same rate at which the display is refreshed). This means that loading 32 into the tone timer would provide a 1 second tone. (32 hex = 50 dec.) The maximum tone duration from a single tone instruction is a little over 5 seconds.

FX15 Load the timer with the contents of the variable indicated by X. Any variable may be used to load the timer with a hex value between 00 and FF. Once the contents of the variable are stored, the interrupt routine handles the counting of the timer down to zero.

FX07 Load the variable indicated by X with the current value of the timer. The contents of the timer, scratchpad loc'n 0020, may be loaded into any variable to check the current value of the timer. This is used to determine when a delay or a "time allowed" period is ended.

Memory Pointer Instructions

AMMM Set the memory address pointer to some memory location 0MMM. This may be used to point to any address from 0000 to 0FFF. Locations over 0FFF cannot be addressed because of the two byte format of CHIP-8 instructions. The instruction is used prior to data transfers between blocks of memory and variables or the display page.

FX1E Increment the memory address pointer by the value of the variable indicated by X. Any variable may be used to increment the memory pointer. This instruction may be used to vary the value of the memory pointer in accordance with some other parameter which may vary during the execution of the program.

FX29 Set the memory address pointer equal to the first byte of the 5 byte display pattern list for the least significant hex digit of the variable indicated by X. The address pointer is set to this location with the intent of displaying the value of the hex digit in the variable indicated. Only the digit derived from the 4 least significant bits of the byte is involved.

FX33 Convert the contents of the variable indicated by X to a 3 digit decimal number and store each digit in separate successive bytes beginning at the location indicated by the memory address pointer. The hexadecimal value in any variable may be converted to 3 decimal digits occupying the least significant digit of 3 different bytes and stored in memory. The use of 3 decimal digits is derived from the fact that 001_6 through $FF1_6$ corresponds to 000_{10} through 255_{10} . The bytes at the memory locations containing the 3 decimal digits may be transferred to variables using an FX65 instruction. An FX29 instruction must be used for each digit contained in a variable to prepare it for display.

FX55 Transfer the contents of 1 to 16 variables to successive memory locations beginning with the location specified by the memory address pointer. The variables saved will always include V0 through the variable indicated by X. The memory address pointer will have been incremented by X plus 1 when the instruction is completed. This instruction and FX65 may be used to store and recall any number of variables to create multiple sets of variables.

FX65 Transfer the contents of successive memory locations beginning with the location specified by the memory address pointer to 1 to 16 variables. The variables will always include V0 through the variable indicated by X. The memory address pointer will have been incremented by X plus 1 when the instruction is completed. This instruction is the reverse of the FX55 instruction.

Display Instructions

DXYN Transfers the pattern byte list pointed to by the memory address pointer. The number of bytes to be transferred in the pattern byte list is indicated by N. The horizontal bit coordinate of the position at which the pattern is to be displayed is contained in the variable indicated by X. The vertical coordinate is contained in the variable indicated by Y. The pattern is EXCLUSIVE-ORED with the existing display in the area. If any spot in the pattern matches a spot in the existing pattern (a HIT condition) VF will be set to 01. If a pattern is shown a second time at the same coordinates it will cancel or erase itself due to the EXCLUSIVE-OR function. The memory pointer is unchanged at the completion of the instruction.

00E0 The active display page is erased by setting every byte to zero.

Monitor Command Summary

[FN][0] = Memory Modify

To examine and/or deposit data in memory, first key in the 4-digit address at which you want to begin. Then press (FN)(0) or (FN)(FN). The 2-digit data at this address will be shown. To step to subsequent locations, press (FN). To deposit new data at the address shown, key in 2 hex digits (one byte). Note that the next address and its data are displayed immediately upon pressing the 2nd digit of each data entry. To verify correct entry of your data, escape from MEMOD by pressing (RST); re-enter the starting address and get into MEMOD again to check memory contents.

[FN][1] = Tape Load

In order to load a program from tape, first use MEMOD to key in the 2-byte starting and ending addresses of the segment of memory concerned, at 0002 to 0005 (incl.). For example, to load a 512 byte block at 0200: deposit 02, 00, 04, 00 at loc's 0002,3,4,5 resp. Playback the cassette until the leader tone (steady 2400 Hz tone) can be heard; then press (FN)(1). The tape must be in motion and playing the leader tone, before (FN)(1) is keyed; If a tape dropout is encountered on the leader, the load will not be successful.

[FN][2] = Tape Dump

Proceed as for tape load by entering the start and end locations at 0002,3,4,5. Run the cassette deck in record mode for about 5 seconds, (10 sec. if at start of tape) to make a leader. Then press (FN)(2). Note that during tape dump and load operations the display blacks out until the operation is complete.

[FN][3] = Run User Program

To run a CHIP-8 program, first enter C000, then press (FN)(3). If C000 is already being displayed, simply press (FN)(3). To run a machine-code program, enter the starting address of the program, then (FN)(3). At any time, (RST) may be used to regain monitor control.

The CHIPOS Monitor had to be kept down to a bare minimum to fit into the available EPROM space. Consequently, there are a few compromises:

- (1) When keying in a 4-digit address, the display will not respond until all 4 digits are entered; (similarly for 2-digit data entries);
- (2) Only one address and its contents can be shown at a time; it would have been desirable to show 2 successive locations, for verification;
- (3) To escape from the MEMOD function, the CPU must be reset; resulting in all input/output devices being reset (which may or may not matter).

However, once you get used to these minor hindrances, you'll find the monitor easy to use and quite versatile. On the 'plus' side, the keying in of programs is facilitated by the "address auto-increment" feature; i.e. there's no need for an 'ENTER' or 'RETURN' key following each byte. Also, the tape load/dump functions, although they are stripped of all sophistication (like checksums, blocking, formatting, etc), have the useful property of letting you relocate programs and data; i.e. you can load into a different block from where you originally dumped.

HEXADECIMAL AND DECIMAL CONVERSION

From hex: locate each hex digit in its corresponding column position and note the decimal equivalents. Add these to obtain the decimal value.

From decimal: (1) locate the largest decimal value in the table that will fit into the decimal number to be converted, and (2) note its hex equivalent and hex column position. (3) Find the decimal remainder. Repeat the process on this and subsequent remainders.

HEXADECIMAL COLUMNS											
6		5		4		3		2		1	
HEX = DEC		HEX = DEC		HEX = DEC		HEX = DEC		HEX = DEC		HEX = DEC	
0	0	0	0	0	0	0	0	0	0	0	0
1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15
0 1 2 3		4 5 6 7		0 1 2 3		4 5 6 7		0 1 2 3		4 5 6 7	
BYTE				BYTE				BYTE			

TABLE OF CHIP-8 INSTRUCTIONS

STORED CODE	MNEMONIC	DESCRIPTION
1MMM	GOTO MMM	JUMP TO INSTRUCTION AT LOCATION MMM
BMMM	GOTO MMM+VO	COMPUTED GOTO; JUMP TO MMM+VO.
2MMM	DO MMM	DO CHIP-8 SUBROUTINE AT MMM.
00EE	RETURN	RETURN FROM CHIP-8 SUBROUTINE
3XKK	SKF VX=KK	SKIP NEXT INSTR'N IF VX=KK (HEX).
4XKK	SKF VX#KK	SKIP IF VX NOT= KK.
5XY0	SKF VX=VY	SKIP IF VX = VY.
9XY0	SKF VX#VY	SKIP IF VX NOT= VY.
EX9E	SKF VX=KEY	SKIP IF KEY DOWN = VX; NO WAIT.
EXA1	SKF VX#KEY	SKIP IF KEY NOT= VX; NO WAIT.
6XKK	VX=KK	ASSIGN HEX CONSTANT KK TO VARIABLE.
CXKK	VX=RND.KK	GET RANDOM BYTE; AND WITH KK.
7XKK	VX=VX+KK	ADD (2'S COMP.) KK TO VX.
8XY0	VX=VY	COPY VY TO VX.
8XY1	VX=VX!VY	LOGICAL OR VX WITH VY.
8XY2	VX=VX.VY	LOGICAL AND VX WITH VY.
8XY4	VX=VX+VY	ADD VY TO VX; IF RESULT>FF, VF=1.
8XY5	VX=VX-VY	SUBTRACT VY; IF VX<VY, VF=0, ELSE 1.
FX07	VX=TIME	GET CURRENT TIMER VALUE.
FX0A	VX=KEY	INPUT HEX KEYCODE (WAIT FOR KEYDOWN)
FX15	TIME=VX	INITIALIZE TIMER; 01 = 20 MILLISEC.
FX18	TONE=VX	BLEEP FOR 20 X VX MILLISECONDS.
AMMM	I=MMM	SET MEMORY INDEX POINTER TO MMM.
FX1E	I=I+VX	ADD VX TO MEMORY POINTER.
FX29	I=DSP,VX	SET POINTER TO SHOW VX (LS DIGIT).
FX33	MI=DEQ,VX	STORE 3-DIGIT DECIMAL EQUIV. OF VX.
FX55	MI=VO:VX	STORE VO THRU VX AT I; (I=I+X+1).
FX65	VO:VX=MI	LOAD VO THRU VX AT I; (I=I+X+1).
00E0	ERASE	CLEAR THE SCREEN.
OMMM	CALL MMM	CALL MACHINE-CODE SUBR. (MMM>200).
DXYN	SHOW N@VX,VY	DISPLAY N-BYTE PATTERN AT (VX,VY).
0000	NOP	NO OPERATION (DEFAULT INSTR'N).
F000	STOP	JUMP TO MONITOR (CHIPOS); (DEFAULT).

[X,Y,K,N AND M ARE ARBITRARY HEX DIGITS, 0 TO F.]

ACCUMULATOR AND MEMORY		ADDRESSING MODES															BOOLEAN/ARITHMETIC OPERATION						COND. CODE REG.					
		IMMED			DIRECT			INDEX			EXTND			INNER			(All register labels refer to contents)						5 4 3 2 1 0					
		OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#												
OPERATIONS	MNEMONIC																											
Add	ADDA	8B	2	2	9B	3	2	AB	5	2	BB	4	3				A + M → A											
	ADDB	CB	2	2	DB	3	2	EB	5	2	FB	4	3				B + M → B											
Add Acmltrs	ABA													1B	2	1	A + B → A											
Add with Carry	ADCA	89	2	2	99	3	2	A9	5	2	B9	4	3				A + M + C → A											
	ADCB	C9	2	2	D9	3	2	E9	5	2	F9	4	3				B + M + C → B											
And	ANDA	84	2	2	94	3	2	A4	5	2	B4	4	3				A ← M · A											
	ANDB	C4	2	2	D4	3	2	E4	5	2	F4	4	3				B ← M · B											
Bit Test	BITA	85	2	2	95	3	2	A5	5	2	B5	4	3				A ← M											
	BITB	C5	2	2	D5	3	2	E5	5	2	F5	4	3				B ← M											
Clear	CLR							6F	7	2	7F	6	3				00 → M											
	CLRA													4F	2	1	00 → A											
	CLRB													5F	2	1	00 → B											
Compare	CMPA	81	2	2	91	3	2	A1	5	2	B1	4	3				A ← M											
	CMPB	C1	2	2	D1	3	2	E1	5	2	F1	4	3				B ← M											
Compare Acmltrs	CBA													11	2	1	A ← B											
Complement, 1's	COM							63	7	2	73	6	3				M ← M											
	COMA													43	2	1	A ← A											
	COMB													53	2	1	B ← B											
Complement, 2's (Negate)	NEG							60	7	2	70	6	3				00 ← M · M											
	NEGA													40	2	1	00 ← A · A											
	NEGB													50	2	1	00 ← B · B											
Decimal Adjust, A	DAA													19	2	1	Converts Binary Add. of BCD Characters into BCD Format											
Decrement	DEC							6A	7	2	7A	6	3				M ← M - 1											
	DECA													4A	2	1	A ← A - 1											
	DECB													5A	2	1	B ← B - 1											
Exclusive OR	EORA	88	2	2	98	3	2	A8	5	2	B8	4	3				A ← M ⊕ A											
	EORB	C8	2	2	D8	3	2	E8	5	2	F8	4	3				B ← M ⊕ B											
Increment	INC							6C	7	2	7C	6	3				M ← M + 1											
	INCA													4C	2	1	A ← A + 1											
	INCB													5C	2	1	B ← B + 1											
Load Acmltr	LDAA	86	2	2	96	3	2	A6	5	2	B6	4	3				M ← A											
	LDAB	C6	2	2	D6	3	2	E6	5	2	F6	4	3				M ← B											
Or, Inclusive	ORA	8A	2	2	9A	3	2	AA	5	2	BA	4	3				A ← M ∨ A											
	ORB	CA	2	2	DA	3	2	EA	5	2	FA	4	3				B ← M ∨ B											
Push Data	PSHA													36	4	1	A ← M _{SP} , SP ← SP - 1											
	PSHB													37	4	1	B ← M _{SP} , SP ← SP - 1											
Pull Data	PULA													32	4	1	SP ← SP + 1, M _{SP} ← A											
	PULB													33	4	1	SP ← SP + 1, M _{SP} ← B											
Rotate Left	ROL							69	7	2	79	6	3				M ← M											
	ROLA													49	2	1	A ← M											
	ROLB													59	2	1	B ← M											
Rotate Right	ROR							66	7	2	76	6	3				M ← M											
	RORA													46	2	1	A ← M											
	RORB													56	2	1	B ← M											
Shift Left, Arithmetic	ASL							68	7	2	78	6	3				M ← M											
	ASLA													48	2	1	A ← M											
	ASLB													58	2	1	B ← M											
Shift Right, Arithmetic	ASR							67	7	2	77	6	3				M ← M											
	ASRA													47	2	1	A ← M											
	ASRB													57	2	1	B ← M											
Shift Right, Logic	LSR							64	7	2	74	6	3				M ← M											
	LSRA													44	2	1	A ← M											
	LSRB													54	2	1	B ← M											
Store Acmltr	STAA				97	4	2	A7	6	2	B7	5	3				A ← M											
	STAB				D7	4	2	E7	6	2	F7	5	3				B ← M											
Subtract	SUBA	80	2	2	90	3	2	A0	5	2	B0	4	3				A ← M - A											
	SUBB	C0	2	2	D0	3	2	E0	5	2	F0	4	3				B ← M - B											
Subtract Acmltrs	SBA													10	2	1	A ← B - A											
Subtr. with Carry	SBCA	82	2	2	92	3	2	A2	5	2	B2	4	3				A ← M - C + A											
	SBCB	C2	2	2	D2	3	2	E2	5	2	F2	4	3				B ← M - C + B											
Transfer Acmltrs	TAB													16	2	1	A ← B											
	TBA													17	2	1	B ← A											
Test, Zero or Minus	TST							6D	7	2	7D	6	3				M ← 00											
	TSTA													4D	2	1	A ← 00											
	TSTB													5D	2	1	B ← 00											

FIGURE 1 MC6800 Instruction Set

INDEX REGISTER AND STACK		IMMED			DIRECT			INDEX			EXTND			INHER			BOOLEAN/ARITHMETIC OPERATION					
POINTER OPERATIONS	MNEMONIC	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	H	I	N	Z	V	C
Compare Index Reg	CPX	8C		3	9C		4	AC		6	BC		5									
Decrement Index Reg	DEX													09	4	1						
Decrement Stack Pntr	DES													34	4	1						
Increment Index Reg	INX													08	4	1						
Increment Stack Pntr	INS													31	4	1						
Load Index Reg	LOX	CE	3	3	DE	4	2	EE	6	2	FE	5	3									
Load Stack Pntr	LDS	BE	3	3	9E	4	2	AE	6	2	BE	5	3									
Store Index Reg	STX				DF	5	2	EF	7	2	FF	6	3									
Store Stack Pntr	STS				9F	5	2	AF	7	2	BF	6	3									
Indx Reg → Stack Pntr	TXS													35	4	1						
Stack Pntr → Indx Reg	TSX													30	4	1						

JUMP AND BRANCH OPERATIONS		RELATIVE			INDEX			EXTND			INHER			BRANCH TEST						
	MNEMONIC	OP	~	#	OP	~	#	OP	~	#	OP	~	#		5	4	3	2	1	0
															H	I	N	Z	V	C
Branch Always	BRA	20	4	2										None						
Branch If Carry Clear	BCC	24	4	2										C = 0						
Branch If Carry Set	BCS	25	4	2										C = 1						
Branch If = Zero	BEQ	27	4	2										Z = 1						
Branch If ≥ Zero	BGE	2C	4	2										N + V = 0						
Branch If > Zero	BGT	2E	4	2										Z + (N + V) = 0						
Branch If Higher	BHI	22	4	2										C + Z = 0						
Branch If ≤ Zero	BLE	2F	4	2										Z + (N + V) = 1						
Branch If Lower Or Same	BLS	23	4	2										C + Z = 1						
Branch If = Zero	BLT	2D	4	2										N + V = 1						
Branch If Minus	BMI	2B	4	2										N = 1						
Branch If Not Equal Zero	BNE	26	4	2										Z = 0						
Branch If Overflow Clear	BVC	28	4	2										V = 0						
Branch If Overflow Set	BVS	29	4	2										V = 1						
Branch If Plus	BPL	2A	4	2										N = 0						
Branch To Subroutine	BSR	8D	8	2																
Jump	JMP				6E	4	2	7E	3	3				See Special Operations						
Jump To Subroutine	JSR				AD	8	2	BD	9	3										
No Operation	NOP										01	2	1	Advances Prog. Cntr. Only						
Return From Interrupt	RTI										3B	10	1							
Return From Subroutine	RTS										39	5	1	See special Operations						
Software Interrupt	SWI										3F	12	1							
Wait for Interrupt	WAI										3E	9	1							

CONDITIONS CODE REGISTER		INHER			BOOLEAN OPERATION					
OPERATIONS	MNEMONIC	OP	~	#						
Clear Carry	CLC	0C		2	0	C				R
Clear Interrupt Mask	CLI	0E		2	0	I				
Clear Overflow	CLV	0A		2	0	V				
Set Carry	SEC	0D		2	1	C				S
Set Interrupt Mask	SEI	0F		2	1	I				
Set Overflow	SEV	0B		2	1	V				
Accmltr A $\leftarrow$ CCR	TAP	06		2	1	A	CCR			
CCR $\leftarrow$ Accmltr A	TPA	07		2	1	CCR	A			

CONDITION CODE REGISTER NOTES:

- (Bit set if test is true and cleared otherwise)
- (Bit V) Test: Result = 10000000?
 - (Bit C) Test: Result = 00000000?
 - (Bit C) Test: Decimal value of most significant BCD Character greater than nine? (Not cleared if previously set)
 - (Bit V) Test: Operand = 10000000 prior to execution?
 - (Bit V) Test: Operand = 01111111 prior to execution?
 - (Bit V) Test: Set equal to result of $N + C$ after shift has occurred
 - (Bit N) Test: Sign bit of most significant (MS) byte of result = 1?
 - (Bit V) Test: 2's complement overflow from subtraction of LS bytes?
 - (Bit N) Test: Result less than zero? (Bit 15 = 1)
 - (All) Load Condition Code Register from Stack (See Special Operations)
 - (Bit I) Set when interrupt occurs. If previously set, a Non Maskable Interrupt is required to exit the wait state
 - (ALL) Set according to the contents of Accumulator A

LEGEND:

- 00 Byte = Zero.
- OP Operation Code (Hexadecimal).
- ~ Number of MPU Cycles.
- # Number of Program Bytes.
- + Arithmetic Plus.
- Arithmetic Minus.
- * Boolean AND.
- MSP Contents of memory location pointed to be Stack Pointer.
- + Boolean Inclusive OR.
- * Boolean Exclusive OR.
- M Complement of M.
- + Transfer Into.
- 0 Bit = Zero.
- H Half carry from bit 3.
- I Interrupt mask
- N Negative (sign bit)
- Z Zero (byte)
- V Overflow, 2's complement
- C Carry from bit 7
- R Reset Always
- S Set Always
- 1 Test and set if true, cleared otherwise
- Not Affected
- CCR Condition Code Register
- LS Least Significant
- MS Most Significant

FIGURE 1 (continued)

Determine Active CA1 (CB1) Transition for Setting Interrupt Flag IRQA(B)1 – (bit b7)

- b1 = 0 : IRQA(B)1 set by high-to-low transition on CA1 (CB1).
b1 = 1 : IRQA(B)1 set by low-to-high transition on CA1 (CB1).

IRQA(B) 1 Interrupt Flag (bit b7)

Goes high on active transition of CA1 (CB1); Automatically cleared by MPU Read of Output Register A(B). May also be cleared by hardware Reset.

CA1 (CB1) Interrupt Request Enable/Disable

- b0 = 0 : Disables IRQA(B) MPU Interrupt by CA1 (CB1) active transition.¹
b0 = 1 : Enable IRQA(B) MPU Interrupt by CA1 (CB1) active transition.
1. IRQA(B) will occur on next (MPU generated) positive transition of b0 if CA1 (CB1) active transition occurred while interrupt was disabled.

b7	b6	b5	b4	b3	b2	b1	b0
IRQA(B)1 Flag	IRQA(B)2 Flag	CA2(CB2) Control			DDR Access	CA1(CB1) Control	

IRQA(B)2 Interrupt Flag (bit b6)

CA2 (CB2) Established as Input (b5 = 0): Goes high on active transition of CA2 (CB2); Automatically cleared by MPU Read of Output Register A(B). May also be cleared by hardware Reset.

CA2 (CB2) Established as Output (b5 = 1): IRQA(B)2 = 0, not affected by CA2 (CB2) transitions.

Determines Whether Data Direction Register Or Output Register is Addressed

- b2 = 0 : Data Direction Register selected.
b2 = 1 : Output Register selected.

CA2 (CB2) Established as Output by b5 = 1

(Note that operation of CA2 and CB2 output functions are not identical)

b5 b4 b3
1 0 → CA2

b3 = 0 : Read Strobe With CA1 Restore
CA2 goes low on first high-to-low E transition following an MPU Read of Output Register A; returned high by next active CA1 transition.

b3 = 1 : Read Strobe with E Restore
CA2 goes low on first high-to-low E transition following an MPU Read of Output Register A; returned high by next high-to-low E transition.

→ CB2

b3 = 0 : Write Strobe With CB1 Restore
CB2 goes low on first low-to-high E transition following an MPU Write into Output Register B; returned high by the next active CB1 transition.

b3 = 1 : Write Strobe With E Restore
CB2 goes low on first low-to-high E transition following an MPU Write into Output Register B; returned high by the next low-to-high E transition.

b5 b4 b3
1 1 → Set/Reset CA2 (CB2)

CA2 (CB2) goes low as MPU writes b3 = 0 into Control Register.
CA2 (CB2) goes high as MPU writes b3 = 1 into Control Register.

CA2 (CB2) Established as Input by b5 = 0

b5 b4 b3
0 → CA2 (CB2) Interrupt Request Enable/Disable

b3 = 0 : Disables IRQA(B) MPU Interrupt by CA2 (CB2) active transition.¹
b3 = 1 : Enables IRQA(B) MPU Interrupt by CA2 (CB2) active transition.
1. IRQA(B) will occur on next (MPU generated) positive transition of b3 if CA2 (CB2) active transition occurred while interrupt was disabled.

→ Determines Active CA2 (CB2) Transition for Setting Interrupt Flag IRQA(B)2 – (bit b6)

b4 = 0 : IRQA(B)2 set by high-to-low transition on CA2 (CB2).
b4 = 1 : IRQA(B)2 set by low-to-high transition on CA2 (CB2).

FIGURE 2. PIA Control Register Format

